My notes on LLM

Tri Nguyen

nguyetr9@oregonstate.edu

August 25, 2023

1 Why Attention/Transformer is effective?

There are many criteria to start with, but I want to focus solely on model performance. Having said that, many article and even the Transformer paper Vaswani et al. 2017 mentioned that parallelism is a big advantage it brings to the table. To my view, that's a somewhat secondary concern. If you can't run it using 1 GPU, then run it with 10 GPUs, as long as it brings significant improvement.

Instead, our question here lies on the architecture, or the parameterization strategy. In particular, I'm interesting in the expressiveness of the model.

As we are dealing with sequential data, a data point would have this structure: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, where $\boldsymbol{x}_i \in \mathbb{R}^{d_1}$ is a feature vector representing the i-th step, and n is the sequence length. The general scheme when designing the architecture looks like this:

$$[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \quad \rightarrow \quad [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n] \quad \rightarrow \quad \boldsymbol{y},$$

where $\boldsymbol{y}_i$ is a corresponding hidden vector for the i-step, the collection of $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ can be though of hidden representation of the input sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. Then depending on the final task, many will aggregate hidden representation to a fixed-dimensional vector $\boldsymbol{y} \in \mathbb{R}^{d_2}$. Examples for this scheme includes almost every papers [x, x, x], ranging from, in terms of architecture RNN, LSTM, Transformer; in terms of applications translation, classification, language model, text generation, ...

So we will follow this general scheme and try to design and compare the expressiveness roughly for every choice.

Option 1. As naive as I am, why don't think of input as a matrix $\boldsymbol{X} \in \mathbb{R}^{d_1 \times n}$, then the hidden $\boldsymbol{Y}$ is

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W},$$

where $\boldsymbol{W}$ is the learnable weight. If we take a closer look, the $\boldsymbol{y}_i$ equals to

$$\boldsymbol{y}_j = \sum_{i=1}^{n} \boldsymbol{x}_i W_{j,i}.$$

This means that the hidden vector for the i-step is a linear combination of all the steps input.

- Depending on applications, we can restrict the range of input i, such that $\boldsymbol{y}_i$ should not be depend on the future steps $i+1, i+2, \ldots, n$. This issue can be fixed easily.
- What about non-linearity? We can have a nonlinear function such as σ on top of that, i.e,

  $$\boldsymbol{Y} = \sigma(\boldsymbol{W}\boldsymbol{X}),$$

  where σ is a element-wise nonlinear function, such as ReLU, sigmoid, or tanh. The use of element-wise nonlinear function is because this is a very common piece used to build the deep network. Otherwise, I don't know any other options.

And of course, as an obvious option, we can make this deeper with multiple layers.

I don't see any big issue with this parameterization.

Option 2. RNN/LSTM Generally, RNN/LSTM parameterization has this structure:

$$\begin{aligned}\boldsymbol{s}_1 &= \boldsymbol{f}(\boldsymbol{s}_0)\\ \boldsymbol{s}_2 &= \boldsymbol{f}(\boldsymbol{s}_1, \boldsymbol{x}_1)\\ \boldsymbol{y}_1 &= \boldsymbol{g}(\boldsymbol{s}_0, \boldsymbol{x}_1)\\ \boldsymbol{y}_2 &= \boldsymbol{g}(\boldsymbol{s}_1, \boldsymbol{x}_2)\end{aligned}$$

and generally,

$$\begin{aligned}\boldsymbol{s}_i &= \boldsymbol{f}(\boldsymbol{s}_{i-1}, \boldsymbol{x}_{i-1})\\ \boldsymbol{y}_i &= \boldsymbol{g}(\boldsymbol{s}_{i-1}, \boldsymbol{x}_i)\end{aligned}$$

The most important piece of RNN/LSTM is that it has the state representation $\boldsymbol{s}_i$. The state $\boldsymbol{s}_i$ has the recurrent relation to previous states. And the hidden representation at step $i+1$ depends on 2 factors: the feature at step $i+1$ and the state vector $\boldsymbol{s}_i$. It looks quite intuitive. However, I'd like to argue that in terms of expressiveness, it is even worse than the native option 1. The reason is its structure inherently a Markov chain, i.e, hidden representation at step i only depends on all the input at step $i-1$ (state $\boldsymbol{s}_{i-1}$ and input at step i), nothing else.

This significantly hinders the ability to express long-term dependency. For example, suppose that there are 2 sequences x_1, x_2, x_3 and x'_1, x_2, x_3 both producing the same state s_3 the third step. Then the next hidden representation y_4 would also be the same for both input sequences, regardless of the fact that x_1 and x'_1 might have affect y_4.

I think this is a very big obstacle for RNN/LSTM to learn long term dependency. Not that our naive model does not suffer from this. Every hidden y_i depends directly on every x_j. This blog [1] gives a very nice example:

```
Check the program log and find out whether it ran please.
Check the battery log and find out whether it ran down please.
```

Option 3. The dilated CNN. The dilated CNN takes certain motivation from CNN, of course. But its biggest motivation is about computation though, in comparison to RNN/LSTM. In particular, it wants to avoid the sequential computation of RNN/LSTM while retains the capability of learning long term dependency.

If we look at our first option, we have a fully connect layer to parameterize the dependence of i on every $j < i$. CNN does the same thing but instead of have just one $\boldsymbol{W}$ to learn that long range of dependence, it has multiple filters, each tries to learn a particular pattern, with possibly different range. The thing is, as mentioned nicely here[2], is that to learn all possible patterns of length r, it needs e^r number filters, each filter responses for one pattern. This is not a issue with image data since the data size is fixed, hence the filter size is fixed, not varying as sequential input like text.

So this model, is better than RNN/LSTM in terms of expressing the direct dependency of i, j as same as opt 1, is a bit of limited in terms of the length of dependency. So my suspicion is that it should be bettern RNN/LSTM if the range of dependency is not too large.

Option 4. Okay, what else can we do? Back to option 1, think of each $\boldsymbol{W}$ one particular pattern of dependency of max length (sequence length). We can use multiple $\boldsymbol{W}$s to enhance our model. But still, how many $\boldsymbol{W}$ is enough? Exponentially many. For the example above, we need a dependency of length 8? Haha, but it is not truly a dependency of length 8, isn't? It is a dependency between 2 words 6 words away from each other. So that is the dependency of 'length' 2 only.

And oh man, that is the core idea of attention where I rarely see people pointing it out. How to make the i step depends "directly" on 2 other steps. Well, that is inherently a second-order statistic:

$$\boldsymbol{y}_i \text{ should depends on all possible combinations of } \boldsymbol{x}_i \text{ and } \boldsymbol{x}_j,\ \forall j < i$$

[1] `https://e2eml.school/transformers.html`

[2] `https://ai.stackexchange.com/a/20084`

That is our missing piece in option 1. To realize that, we can model

$$\boldsymbol{y}_i = \boldsymbol{f}(\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_1), \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_2), \ldots, \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_{i-1}))$$

In comparison,

- Option 1: $\boldsymbol{y}_i = \boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_i)$. This is similar to a comparison between fully connected layer vs CNN.

In particular, in the option 1, we parameterize the dependence of $\boldsymbol{y}_i$ on a pattern of length 1, while it is a pattern of length 2, and the pattern is not necessarily a list of consecutive words.

So in that regards, we can extend model capacity by

- Creating multiple patterns. This is corresponding to multiple filters as in CNN, instead of 1 fixed, non-learnable as a dot product used in Transformer. This might be the idea of multiple heads using in Transformer.
- Parameterizing a pattern of length 3, instead of length 2.
- Using certain order-sensitive functions for the $\boldsymbol{g}$ function.

Not everything is about model capacity, isn't? It is also about introducing the right bias to the model.